MP090284

MITRE PRODUCT

Cursor-on-Target Message Router User's Guide

November 2009

Michael J. Kristan
Jeffrey T. Hamalainen
Douglas P. Robbins
Patrick J. Newell

Sponsor:	ESC	**Contract No.:**	FA8721-10-C-0001
Dept. No.:	E146	**Project No.:**	031A3MA0-CT

MITRE

202 Burlington Road
Bedford, Massachusetts 01730

Table of Contents

List of Figures

List of Tables

1 Document Goal

This document describes functionality of the MITRE developed, proof of concept message router named the Cursor-on-Target Message Router. The goal is to provide sufficient detail to understand its intended use and to facilitate a general development of a test strategy for Cursor-on-Target (CoT). The discussion in this document assumes some familiarity with the CoT concepts, schema, and sub-schema. The next sections include background information on CoT and Regular Expressions for reference purposes. More information about CoT can be found at cot.mitre.org. The remainder of the document demonstrates configuration of CoT message subscriptions, examples, and some more advanced uses.

2 Cursor-on-Target Information

The Cursor-On-Target (CoT) data strategy centers on the use of a "common language" for tactical systems that is critical in communicating much needed time sensitive position information. Analogous to functioning acceptably in foreign countries, while only learning a few important words of the native language, CoT starts with a focus on a particular set of important common information on the battlefield. This is seen as a time sensitive position or the "What, When, and Where" (W3) of a specific event. The proof of concept prototype also allows for structured special purpose extensions.

2.1 CoT Base Schema

The CoT "base" schema defined in "Event.xsd," is registered in the DISA DoD XML registry and available at the website cot.mitre.org. It defines a terse schema for representing W3 information, with a total of 12 mandatory fields as outlined in Figure 2-1 below, with a compliant example message in Table 2-1. The CoT base schema is terse by design, but also defines a mechanism for extension. The contents of CoT base schema were selected after studying a large number of DoD systems and realizing that many of the information exchange needs between these systems had a common core set, W3.

```
<?xml version='1.0' standalone='yes'?>
<event version="2.0"
    uid="J-01334"
    type="a-h-A-M-F-U-M"
    time="2005-04-05T11:43:38.07Z"
    start="2005-04-05T11:43:38.07Z"
    stale="2005-04-05T11:45:38.07Z" >
  <detail>
  </detail>
  <point lat="30.0090027" lon="-85.9578735" ce="45.3"
              hae="-42.6" le="99.5"  />
</event>
```

Figure 2-1. Basic CoT XML Message

Table 2-1. Required CoT Fields

Element	Attribute	Opt/ Req	Definition	XML Schema Type
Event	version	Req	Schema version of this event instance (e.g. 2.0)	Decimal equal to 2.0
	type	Req	Hierarchically organized hint about event type	string of pattern "\w+(-\w+)*(;[^;]*)?"
	uid	Req	Globally unique name for this information on this event	string
	time	Req	time stamp: when the event was generated	dateTime
	start	Req	starting time when an event should be considered valid	dateTime
	stale	Req	ending time when an event should no longer be considered valid	dateTime
	how	Req	Gives a hint about how the coordinates were generated	string of pattern "\w-\w"
	opex	Opt		
	qos	Opt		
	access	Opt		
Point	lat	Req	Latitude referred to the WGS 84 ellipsoid in degrees	decimal -90 to 90 inclusive
	lon	Req	Longitude referred to the WGS 84 in degrees	decimal -180 to 180 inclusive
	hae	Req	Height above the WGS ellipsoid in meters	decimal
	ce	Req	Circular 1-sigma <u>or</u> a circular area about the point in meters	decimal
	le	Req	Linear 1-sigma error <u>or</u> an altitude range about the point in meters	decimal
Detail	N/A	Opt	An optional element used to hold CoT sub-schema.	empty element

2.2 Cot Sub-Schema

The CoT approach allows for definition of sub-schema based on common information exchange needs between systems that extend beyond the information provided by the basic W3 schema. One or more sub-schema may appear in the "detail" element in the base schema. This approach provides additional information exchange capability to the various systems that need to exchange more than the common schema allows. For example, community applications that need W3 data exchanges to meet mission requirements may also need to know about the velocity of an event as well. If the velocity information was added to extend the base schema, CoT would have to define "no statement" values for these fields when they are not used, or make them optional. This will add complexity to software development projects that use the common W3 schema. CoT sub-schemas provide a means to extend CoT without the need for special-case processing. Other advantages to this approach include the encapsulation of data definition in small, manageable "standards," components. This hierarchical extension mechanism allows for the representation of varying degrees of detailed information, and acts as a natural filter mechanism for specific details based on one of many factors that includes access, security, and bandwidth limitations. At the time this document was authored, there are 12 defined sub-schemas. Of these 12, the 4 outlined in Table 2-2 are used by the MTCD Plug-In adapter. All sub-schema definitions can be viewed or downloaded from the cot.mitre.org website.

Table 2.2. CoT Sub-Schema Examples

Sub-Schema	XSD Name	Description
track	CoT_track.xsd	Velocity vector information
flow-tags	CoT__flow-tags_.xsd	Time stamped "fingerprints" for systems which have touched a CoT event. Used for work flow and routing decisions for CoT messages.
uid	CoT_uid.xsd	Provides a place to annotate a CoT message with the unique identifier used by a particular system. (eg. add the TJ track number)
remarks	CoT_remarks.xsd	Provides a place to annotate CoT with free text information.

2.3 CoT *Type* Field Details

The Cursor-on-Target Message Router often uses the CoT *type* string as a test for publication. For this reason, the *type* field is briefly explained here.

The set of possible CoT types is defined by a tree structure. The *type* attribute, as defined in Event.xsd, identifies a specific node in the type tree. It is a hyphen delimited set of alpha-numeric characters. For example, a-f-G represents a type tree of "atoms-friendly-ground." The root element (first character in the type field) of the tree has several values to include atoms, bits, reservations, capability descriptions, etc. Figure 2-3 shows a diagram of valid atomic events.

$$\text{Valid Atomic Event Combinations} =$$

$$a\,(\text{atom}) - \left\{ \begin{array}{c} p\ (\text{pending}) \\ u\ (\text{unknown}) \\ a\ (\text{assumed friend}) \\ f\ (\text{friend}) \\ n\ (\text{neutral}) \\ s\ (\text{suspect}) \\ h\ (\text{hostile}) \\ j\ (\text{joker}) \\ k\ (\text{faker}) \\ o\ (\text{none specified}) \\ x\ (\text{other}) \end{array} \right. - \left\{ \begin{array}{c} P\ (\text{Space}) \\ A\ (\text{Air}) \\ G\ \text{Ground)} \\ S\ (\text{Sea Surface}) \\ U\ (\text{Sea Subsurface}) \\ X\ (\text{Other}) \end{array} \right. - [\text{MIL-STD-2525 Function Code(s)}]$$

Figure 2.3. Valid Atomic Event Combinations

The atoms branch of the tree leverages the MIL-STD-2525B specification for defining the detailed type. The string is broken out as atoms-affiliation-battle dimension from 2525-function code from 2525. For example, the type string "a-h-A-M-A" represents "atoms-hostile-Airborne-Military-Attack/Strike." The lower case a-h prefix is defined in Event.xsd. The first -A- is the MIL-STD-2525 battle dimension (air, ground, surface, etc). The remaining characters are the MIL-STD-2525 function code. In this example, M-A represents "Military-Attack/Strike." In the discussion found below, the "atoms" branch of the type string will be described by the term's affiliation, battle dimension, and 2525 function code per the description in this paragraph. For more information on the other root elements, see the cot.mitre.org website.

3 Regular Expressions

Regular Expressions (Reg Exps) are a special syntax for rule-based tests on strings of characters. Reg Exps form logic to check strings for specific character patterns. A Reg Exp, for instance, could check a string to see if it was “foo”, or if it started with a digit. Regular expressions can also be strung together to form very complex chains, for instance testing for a string that starts with three non-digits, followed by a vowel, and ends in no more than seven blank spaces. A comprehensive list of regular expressions can be found in Table 3-1, which is followed by examples.

(The following table and examples are from
http://aspn.activestate.com/ASPN/docs/ActivePerl/lib/Pod/perlintro.html)

Some brief examples:

^\d+	string starts with one or more digits
^$	nothing in the string (start and end are adjacent)
(\d\s){3}	three digits, each followed by a whitespace character (eg "3 4 5 ")
(a.)+	matches a string in which every odd-numbered letter is ‘a’ (eg "abacadaf")
(\D){3}[aeiou](\s{0,7})$	three non-digits, followed by a vowel and ending with no more than seven blank spaces.

Table 3-1. Regular Expressions

Reg Exp	Description
.	a single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit
\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo|bar|baz)	matches any of the alternatives specified
^	start of string
$	end of string
Quantifiers can be used to specify how many of the previous thing you want to match on, where ``thing" means either a literal character, one of the metacharacters listed above, or a group of characters or metacharacters in parentheses.	
*	zero or more of the previous thing
+	one or more of the previous thing
?	zero or one of the previous thing
{3}	matches exactly 3 of the previous thing
{3,6}	matches between 3 and 6 of the previous thing
{3,}	matches 3 or more of the previous thing

4 Additional Information

For additional information, please see our web site: http://cot.mitre.org.

5 Cursor-on-Target Message Router Overview

The Cursor-on-Target Message Router will route CoT messages from one host to another. The routing is rule-based and provides a one-to-one or one-to-many route capability. A subscription is set up in the Cursor-on-Target Message Router that contains the routing rules and other pertinent information, such as destination address. Multiple subscriptions can be created and set to filter messages differently and reroute information to different hosts.

The rules are broken into two types of test: spatial-temporal bounds and regular expression. Spatial-temporal bounds refer to the CoT message location, or "point." A matching message would have its latitude, longitude, and height within the bounds of the subscriptions test. Regular expressions are more varied tests that can be specially tailored to check any CoT attribute.

In addition to routing, messages can be modified before transmission. Also, all incoming messages are cached for future use. This allows new subscriptions to reroute matching messages that are still active. Caching occurs before modification.

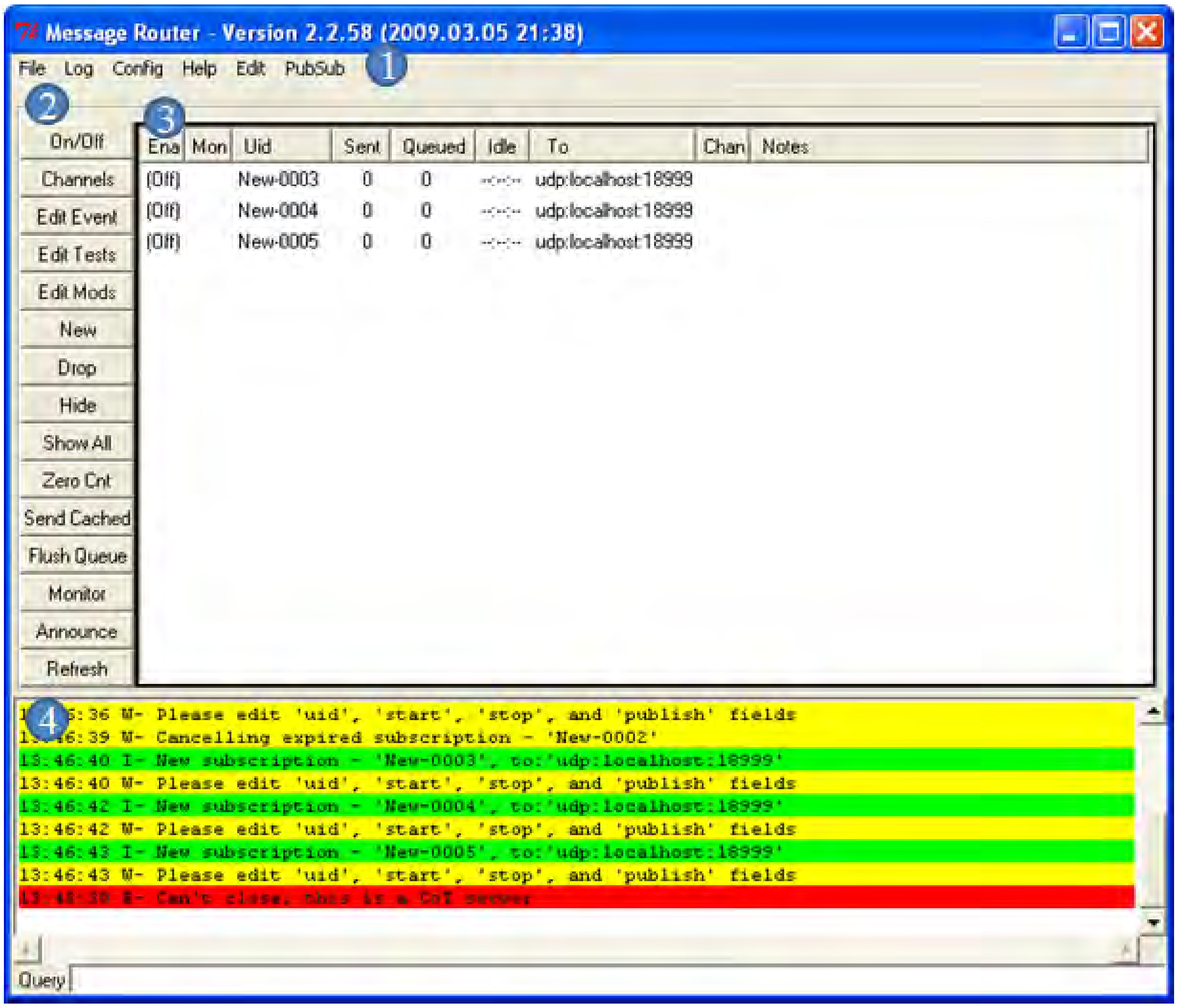
The Cursor-on-Target Message Router is a stand-alone application that can be installed on any Windows or Linux machine, provided it has a serial or network connection to other CoT applications.

6 General Operation

This section is designed to highlight the necessary features required to set up a basic message subscription. While this isn't intended as a complete user's guide, it should contain sufficient detail to set up and begin using the Cursor-on-Target Message Router for testing purposes.

6.1 Graphical User Interface

Figure 6-1 shows the Cursor-on-Target Message Router message router's graphical user interface (GUI). On it there will be circled numbers pointing to specific areas of interest. Each of these numbers corresponds to the following subsections of 6-1.



Ena	Mon	Uid	Sent	Queued	Idle	To	Chan	Notes
(Off)		New-0003	0	0	--:--:--	udp:localhost:18999		
(Off)		New-0004	0	0	--:--:--	udp:localhost:18999		
(Off)		New-0005	0	0	--:--:--	udp:localhost:18999		

```
1[illegible]6:36 W- Please edit 'uid', 'start', 'stop', and 'publish' fields
1[illegible]46:39 W- Cancelling expired subscription - 'New-0002'
13:46:40 I- New subscription - 'New-0003', to:'udp:localhost:18999'
13:46:40 W- Please edit 'uid', 'start', 'stop', and 'publish' fields
13:46:42 I- New subscription - 'New-0004', to:'udp:localhost:18999'
13:46:42 W- Please edit 'uid', 'start', 'stop', and 'publish' fields
13:46:43 I- New subscription - 'New-0005', to:'udp:localhost:18999'
13:46:43 W- Please edit 'uid', 'start', 'stop', and 'publish' fields
13:48:30 E- Can't close, this is a CoT server
```

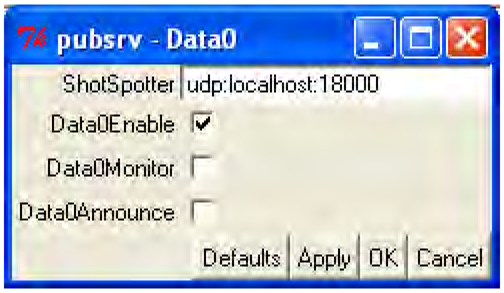
Figure 6-1. Message Router Graphical User Interface

6.1.1 PubSub (1)

There are three options under the PubSub pull-down menu; the one of importance to us now is “Data0”. The ‘0’ specifies the channel number. The use of channels will be discussed in section 6.2. Selecting “Data0” will launch a small window, seen in Figure 6-2. There is one data entry block and three checkboxes. The data entry block, which in this example is labeled, “ShotSpotter”, is used to determine where the message router listens for incoming messages. The format of this is: “Protocol:Host:Port.” The Cursor-on-Target Message Router supports a few protocols, the most common being udp and tcp. Examples of these include:

1. udp:localhost:19000
2. tcp:192.0.0.1:17569

The first checkbox, labeled “Data0Enable”, will enable listening on this channel when selected. The next checkbox, “Data0Monitor”, will enable monitoring of this channel when selected. The last checkbox, “Data0Announce”, will announce activity of this channel to the log window when selected.



ShotSpotter udp:localhost:18000
Data0Enable ✓
Data0Monitor
Data0Announce

Figure 6-2. PubSub Data Window

6.1.2 Active Subscription Buttons (2)

The most important button is ‘New,’ which creates a new, generic subscription. The representation of this is seen in the Active Subscription Window (see section 6.1.3 for more details). This subscription has no routing rules or modifications associated with it. The ‘Edit Event, ‘Edit Tests,’ and ‘Edit Mods’ buttons are used to change the subscription to route specific messages to the desired host.

Most buttons require one or more subscriptions to be highlighted in order to work. Once selected, these buttons are used to modify the characteristics of that subscription. Some other buttons, namely ‘Show All’ and ‘Refresh,’ perform general operations on all of the subscriptions. The names and functions of each button are listed below in Table 6-1.

Table 6-1. Active Subscription Buttons and Functions

Name	Function
On/Off	Toggles the subscription between off and on.
Channels	Pops up a window that allows you to manage channels.
Edit Event	Pops up a window that allows you to change basic information. See section 5.2 for more info.
Edit Tests	Pops up a window that allows you to set tests for rerouting messages. See section 5.3 for more info.
Edit Mods	Pops up a window that allows you to set modifications to matching messages. See section 5.4 for more info.
New	Creates a new, generic subscription in the Subscription Window (section 6.1.3).
Drop	Permanently removes the selected subscription.
Hide	Removes the subscription from the Active Subscription window, while remaining active. "…hidden…" will appear in Subscription Window.
Show All	Restores hidden subscriptions to their normal state. Double clicking "…hidden…" produces the same result.
Zero Cnt	Resets the "Sent" count for this subscription to 0 (zero).
Send Cached	Resends all cached messages to the subscription, where matching messages will be resent.
Flush Queue	Deletes any messages in the message queue.
Monitor	Toggles the monitoring option in the "Mon" field, denoted by the 'm' or 'M' characters.
Announce	Toggles the announcing option in the "Mon" field, denoted by the 'a' or 'A' characters.
Refresh	Updates the Subscription Window. Will automatically update every 5 seconds.

6.1.3 Active Subscription Window (3)

This window shows a list of active subscriptions in The Cursor-on-Target Message Router. For each subscription there are eight fields of identifying information. The first is "Ena," short for enabled. This can have two values: "(Off)" and "(On)". While a subscription is off, it will not forward any matching CoT messages. The second field, "Mon," short for monitor, is either blank or 'm' when the subscription is disabled and 'M' when the subscription is enabled. Monitoring a subscription is primarily used in testing, and sends any matching messages to the Test port (PubSub->Test). This field is also used to display whether or not announcing is enabled for the subscription. If announcing is enabled, an 'a' or 'A' will also be shown in this field. It is possible for both monitoring and announcing to be enabled for a subscription, which is denoted by 'ma' or 'MA' in the field. The third field is the user identification, or Uid, which is used to tell subscriptions apart. The fourth field, "Sent," shows how many messages matching this subscription have been rerouted, while the fifth field, "Queued," shows how many messages are waiting to be rerouted. "Idle," the sixth field, shows the time since the last message this subscription rerouted. The seventh field shows the host that the subscription will reroute messages to. The eighth field shows the number of the channel being used. The final field, "Notes," shows general comments about the subscription.

Double clicking a subscription within will create a pop-up window with the full details of the subscription, minus its routing rules or modifications. When an event is highlighted and the 'Hide' button is clicked, the subscription will disappear. A line reading "…hidden…" will appear in the Subscription Window as long as there is at least one hidden subscription. Hidden events can be redisplayed when "…hidden…" is double clicked or the 'Show All' button is pressed.

6.1.4 Log Window (5)

Status updates on subscriptions, received CoT messages, help text, and errors are displayed in the log window. Lines of greater importance are highlighted in green, yellow, and red. Yellow and red indicates that the message is a warning or error, respectively. The contents of the log window are written into a logfile for later retrieval. The log window can be manipulated by the pull-down menu Log. There are seven options in total: Scroll, Mark, Note, Clear, Split, Split & Init, Hide.

6.2 Editing Channels

The Cursor-on-Target Message Router makes use of channels to listen for incoming messages. There can be up to 16 channels for any specified data source. The numbering for these channels starts at 0, which goes up to and includes channel 15. To edit the number of channels being used, select "Edit" from the "Config" pull-down menu, and then select "Channels" from the subsequent "Edit" pull-down menu. Selecting "Channels" will launch a small window, seen in Figure 6-3. The number of channels can be selected from the "DataChannels" drop-down selection box. The number of channels that can be selected are 1, 2, 4, 8, and 16. Clicking "OK" will close the window and temporarily save the number of channels, but the changes will not take effect immediately.

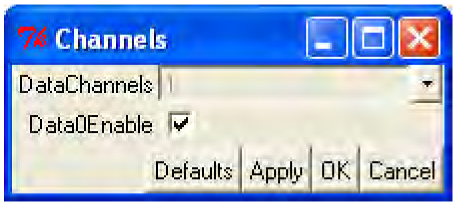
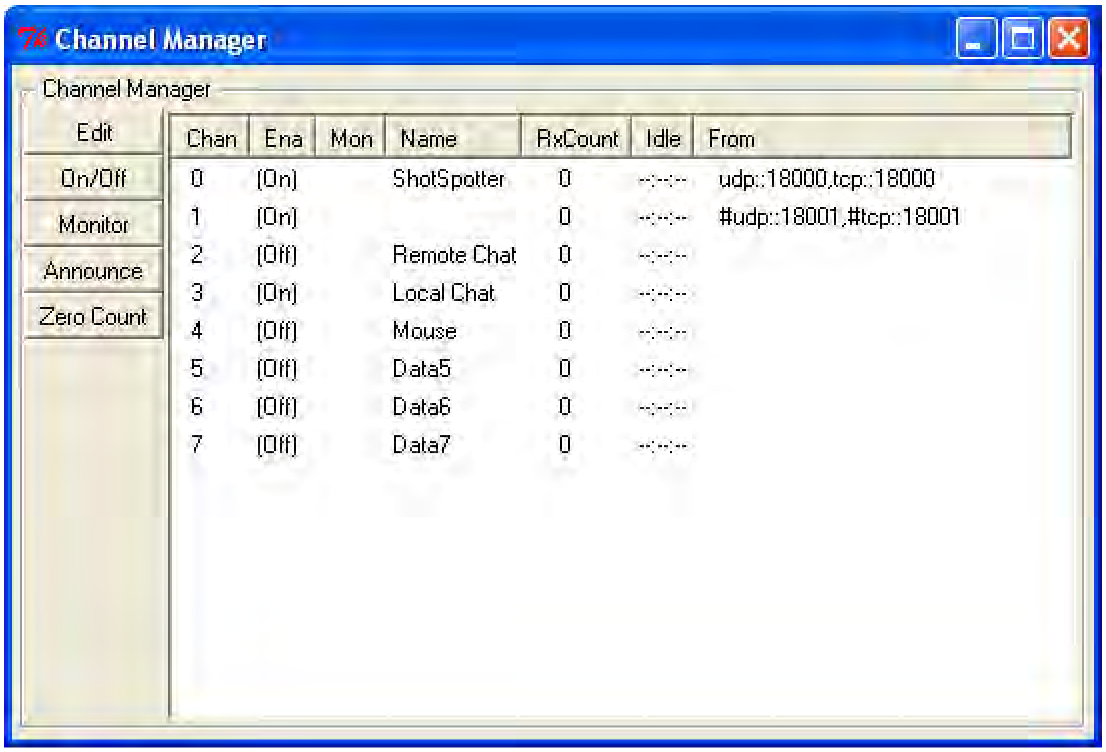


Figure 6-3. Channels Window

In order for these changes to take effect, the configuration must be saved and the Cursor-on-Target Message Router must be restarted. To save the configuration, select “Save” from the “Config” pull-down menu. To restart, select “Exit” from the “File” pull-down menu and start the Cursor-on-Target Message Router once it has closed.

Specific channels can be individually modified through the Channel Manager, shown in Figure 6-4, which can be brought up by selecting the “Channels” button in the Active Subscription Buttons section described in section 6.1.2. The Channel Manager contains five buttons on the left side of the window: Edit, On/Off, Monitor, Announce, and Zero Count. The “Edit” button allows you to modify the name of the selected channel and where that channel listens for incoming messages. The “On/Off” button toggles the channel on and off, which determines whether or not data is received. The “Monitor” and “Announce” buttons toggle monitoring and activity announcements on and off for the channel respectively. The “Zero Count” button resets the “RxCount” field for the selected channel to 0.

The Channel Manager also lists all available channels on the right side of the window, with data for each channel specified in seven fields: Chan, Ena, Mon, Name, RxCount, Idle, and From. The “Chan” field, short for “channel”, specifies the number of the channel, which can range from 0 to 15. The “Ena” field, short for enabled, can have two values: “(Off)” and “(On)”. While a channel is off, it will not receive any messages. The “Mon” field, short for monitor, shows whether or not announcing or monitoring is enabled for the channel. As in the Active Subscription window, announcing is denoted by an ‘a’ or ‘A’ and monitoring is denoted by an ‘m’ or ‘M’. The “Name” field displays the name of the channel. The “RxCount” field displays the number of messages received on the channel. The “Idle” field displays the amount of time that has passed since the last message has been received. The “From” field displays the host from which messages are being received.



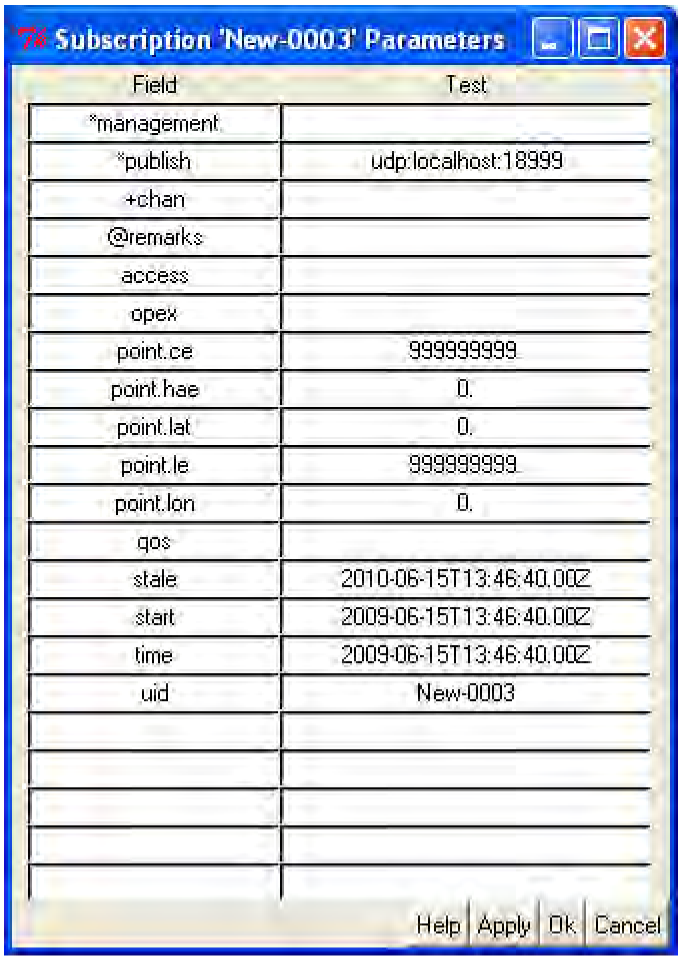
Chan	Ena	Mon	Name	RxCount	Idle	From
0	(On)		ShotSpotter	0	--:--:--	udp::18000,tcp::18000
1	(On)			0	--:--:--	#udp::18001,#tcp::18001
2	(Off)		Remote Chat	0	--:--:--	
3	(On)		Local Chat	0	--:--:--	
4	(Off)		Mouse	0	--:--:--	
5	(Off)		Data5	0	--:--:--	
6	(Off)		Data6	0	--:--:--	
7	(Off)		Data7	0	--:--:--	

Figure 6-4. Channel Manager Window

6.3 Editing Events

To edit a subscription event, that subscription should be highlighted in the Active Subscription Window and the “Edit Event” button must be pressed. The resulting pop-up window can be seen in Figure 6-4. This is the default case. Editing the Publish, Start, Stale, and Uid fields is important regardless of the type of subscription. These are used to determine the publish destination timeframe in which the subscription exists (start and stale), and its name as shown in the Active Subscription Window.

In addition to defining the subscription, a simple routing test can be added that reroutes an incoming CoT event if its position is within the boundary set here. This is done by changing the point fields: ce, hae, lat, le, lon. An event with the same lat, long, and hae will automatically be rerouted, as will any event that is within the range of error (ce and le). As set in the default case, this test will not run, but any field changed will test for that parameter and send only those that pass. Further description can be found in Table 6-2.



Field	Test
*management	
*publish	udp:localhost:18999
+chan	
@remarks	
access	
opex	
point.ce	999999999
point.hae	0.
point.lat	0.
point.le	999999999
point.lon	0.
qos	
stale	2010-06-15T13:46:40.00Z
start	2009-06-15T13:46:40.00Z
time	2009-06-15T13:46:40.00Z
uid	New-0003

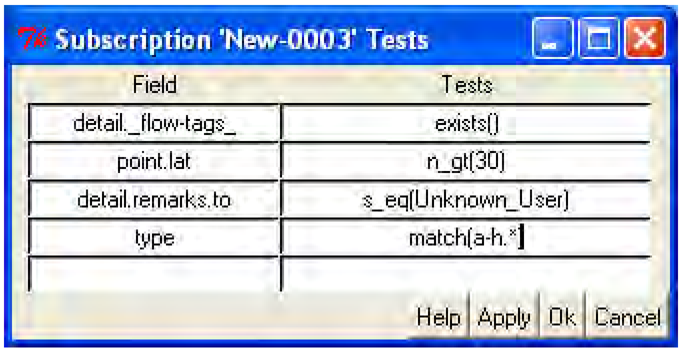
Figure 6-5. Event Parameters Window

Table 6-2. Event Parameters Description

Field	Description	Example
Publish	Host, port, and type connection. Multiple destinations can be listed, separated by a comma.	udp:localhost:10000
Start	Start time for subscription. Defaults with time that subscription was created	2005-02-17T18:38:22.00Z
Stale	This is the stop time for the subscription. Initially set to a year after subscription was started.	2006-02-17T18:38:22.00Z
UID	Display name of subscription	Lat-Lon-Test
Point.lat	Latitude in decimal degrees (North – Plus, South – Minus)	35.0125, -45.1
Point.lon	Longitude in decimal degrees (East – Plus, West – Minus)	123.5, -170.0
Point.hae	Height above ellipsoid (elevation) in meters.	448.4
Point.ce	Circular error radius in meters	25
Point.le	Linear error in meters (height above target)	10

6.4 Editing Tests

The “Edit Test” button instantiates a pop-up window as seen in Figure 6-4. Initially, all fields in the window are blank, though the figure contains examples. The two columns, Field and Tests, are chosen to make very specific tests for rerouting CoT messages. An understanding of CoT XML is required to set-up effective tests. The Field column represents the attributes of the XML (ex. type, stale, etc.). Nested attributes, such as the “lat” attribute hierarchically under “point,” should be separated with a “.” (period) (ex. point.lat, or detail._flow-tags_). The Tests column should be filled with tests that correspond to the Field entry in the same row. For a list of available tests, see Table 6-3. In order for an incoming CoT message to be rerouted to the location specified by the subscription, that message must pass all tests found within the Edit Test window AND any position tests from the Edit Event window. See Section 7 for detailed step-by-step examples.


Subscription 'New-0003' Tests

Field	Tests
detail._flow-tags_	exists()
point.lat	n_gt(30)
detail.remarks.to	s_eq(Unknown_User)
type	match(a-h.*)

Figure 6-6. Edit Test Window

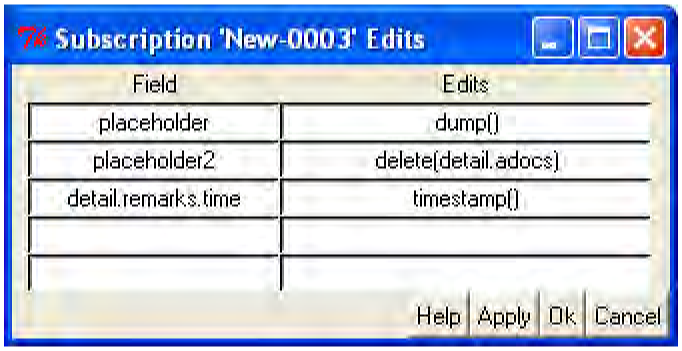
Table 6-3. Edit Test "Tests"

Category	Tests	Description
Test of tag existence	exists	True if field exists in event - 'exists()'
	missing	True if field does not exist in event - 'missing()'
Test if node has certain children	child	True if entity has any children 'detail._flow-tags_=child(ncct,adocs,tadilj)'
	hasany	True if entity has any children 'detail._flow-tags_=hasany(ncct,adocs,tadilj)' Behaves the same way as a child test
	hasnone	True if entity has none of these children 'detail._flow-tags_=hasnone(ncct,adocs,tadilj)'
	hasall	True if entity has all of these children 'detail._flow-tags_=hasall(ncct,adocs,tadilj)"'
Event predicate tests	is	True if event matches any predicate - 'is(neutral,friend)' (see appendix A)
	isall	True if event matches all predicate - 'isall(neutral,ground)'
	isany	True if event matches any predicate - 'isany(bits,tasking)'
	isnot	True if event does not match any predicate - 'isnot(neutral,friend)'
Matching against a field	match	True if field does match regexp - 'match(a-h.*)'
	nomatch	True if field doesn't match regexp - 'nomatch(t-.*)
	Indirect	
Numerical	n_eq	True if event is numerically equal to value - 'n_eq(10)'

Category	Tests	Description
predicate Tests	n_gt	True if field is greater than argument - 'n_gt(27.1)'
	n_in	True if field is in numeric range - 'n_in(100,200)'
	n_lt	True if field is less than argument - 'n_gt(3.14159267)'
	n_out	True if field is out of numerical range - 'n_out(100,200)'
	n_range	True if field is in numeric range - 'n_range(100,200)'
Value in file test	infile	True if field is listed in specified file - 'infile(list.txt)'
	notinfile	True if field is not listed in specified file - 'notinfile(list.txt)'
String predicate tests	s_eq	True if field is lexocographicly equal to arg - 's_gt(dino)'
	s_gt	True if field is lexocographicly greater than arg - 's_gt(dino)'
	s_in	True if field is in lexocographic (string) range - 's_in(betty,wilma)'
	s_lt	True if field is lexocographicly less than arg - 's_gt(dino)'
	s_out	True if field is not in lexocographic (string) range - 's_out(bambam,pebbles)'
	s_range	True if field is in lexocographic (string) range - 's_range(barney,fred)'

6.5 Editing Mods

Pressing the "Edit Mods" button creates the pop-up window seen in Figure 6-5, albeit a new subscription's Edit Test window will be empty. The Field column in Figure 6-7, like the Field column in the Edit Tests window, is where CoT XML attributes are placed. The Edits column will contain the modifications to the associated XML attributes. A list of Edits can be found in Table 6-4. Some Edits don't require specific attributes, but in order for the Edit to work, the associated Field must contain some text, either a random attribute or word (in figure 6-7 the field names placeholder and placeholder2 are used). CoT messages that pass the tests laid out in Edit Tests and Edit Events will have that message modified as described in Edit Mods before it is rerouted. These modifications do affect cached messages.


Subscription 'New-0003' Edits

Field	Edits
placeholder	dump()
placeholder2	delete(detail.adocs)
detail.remarks.time	timestamp()

Figure 6-7. Edit Mods Window

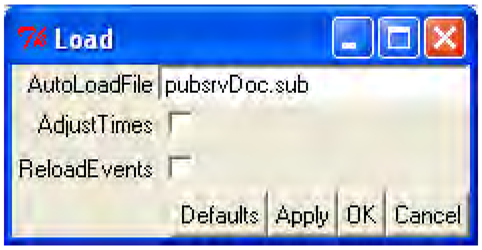
Table 6-4. Edit Mods "Edits"

Edit	Description
delete()	Delete a node and all its children - 'delete(detail.adocs)'
dump()	Show the XML of all events that match this subscription - 'dump()'
rate()	Set the publication rate for this subscription (events per second sent) - 'rate(0.1)'
set()	Set an attribute to some value - 'detail._adocs.foo=set(1234)'
timestamp()	Set this field to a CoT timestamp a number of seconds in the future - 'detail._flow-tags_.test=timestamp(0)'
watch()	Announce all events that match this subscription - 'watch()'

6.6 Managing Subscriptions and Caches

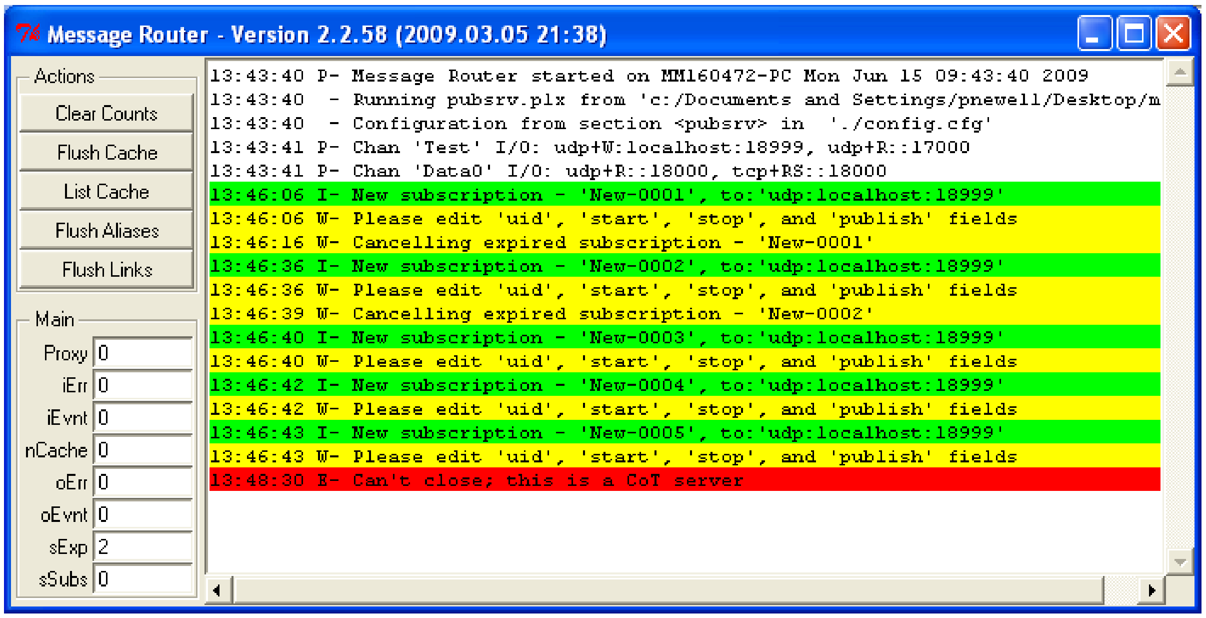
Subscriptions are not persistent between sessions, so they must be saved and reloaded when closing and opening the program, respectively. This is done by selecting the appropriate choice from the pull down menu “File->Save Subs” or “File->Load Subs”. The subscriptions are saved in a “.sub” file using a standard Windows file save dialog box. This file contains the full subscription Event, Tests and Mods. A .sub file can also be automatically loaded by selecting “Config->Edit->Load” from the pull down menu and placing the name of the file into the “AutoLoadFile” entry box as seen in Figure 6-6. When reloaded, these subscriptions will appear the same except for their Sent, Queued, and Idle statistics, which will be reset.

All incoming events are locally cached so that new, or formally un-enabled, subscriptions can retransmit matching events. To clear this cache from the local memory select “File->Debug” from the pull down menu, which will open the window seen in Figure 6-7. This is a larger version of the same Log Window, with five buttons, and a section of statistics. The buttons have the potential to be devastating and should only be used carefully. The first, “Clear Counts”, will zero the number of Sent and Queued events for every subscription. The last button, “Flush Cache”, will remove the cache from memory and cannot be recovered. The statistics reflects the number of incoming and outgoing events, as well as the number of errors, cached events and expired subscriptions (among others).



AutoLoadFile pubsrvDoc.sub
AdjustTimes ☐
ReloadEvents ☐

Figure 6-8. Load Window



```
Proxy   0
iErr    0
iEvnt   0
nCache  0
oErr    0
oEvnt   0
sExp    2
sSubs   0
```

```
13:43:40 P- Message Router started on MM160472-PC Mon Jun 15 09:43:40 2009
13:43:40  - Running pubsrv.plx from 'c:/Documents and Settings/pnewell/Desktop/m
13:43:40  - Configuration from section <pubsrv> in  './config.cfg'
13:43:41 P- Chan 'Test' I/O: udp+W:localhost:18999, udp+R::17000
13:43:41 P- Chan 'Data0' I/O: udp+R::18000, tcp+RS::18000
13:46:06 I- New subscription - 'New-0001', to:'udp:localhost:18999'
13:46:06 W- Please edit 'uid', 'start', 'stop', and 'publish' fields
13:46:16 W- Cancelling expired subscription - 'New-0001'
13:46:36 I- New subscription - 'New-0002', to:'udp:localhost:18999'
13:46:36 W- Please edit 'uid', 'start', 'stop', and 'publish' fields
13:46:39 W- Cancelling expired subscription - 'New-0002'
13:46:40 I- New subscription - 'New-0003', to:'udp:localhost:18999'
13:46:40 W- Please edit 'uid', 'start', 'stop', and 'publish' fields
13:46:42 I- New subscription - 'New-0004', to:'udp:localhost:18999'
13:46:42 W- Please edit 'uid', 'start', 'stop', and 'publish' fields
13:46:43 I- New subscription - 'New-0005', to:'udp:localhost:18999'
13:46:43 W- Please edit 'uid', 'start', 'stop', and 'publish' fields
13:48:30 E- Can't close; this is a CoT server
```

Figure 6-9. Debug Window

7 Subscription Examples

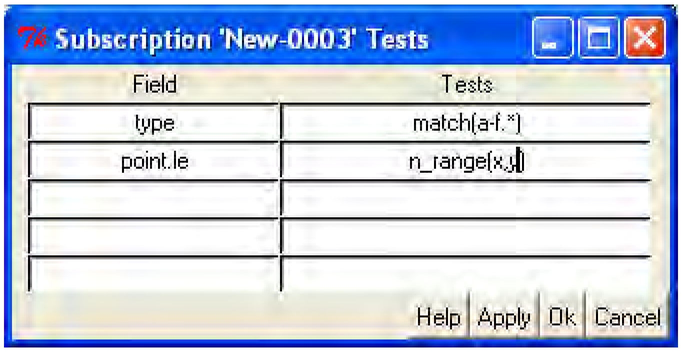
The following examples are designed to illustrate methods for creating subscriptions to filter messages based on different situations. Each describes the tests necessary as well as a rationale/explanation for each decision.

7.1 Blue Force Tracking

Blue force tracking is a way of plotting the location of friendly units on a detailed map and a valuable tool to prevent fratricide during combat. With multiple instances of that program running, one instance may only track friendly units in certain ranges of latitude and longitude. The following tests show you how to reroute only messages dealing with friendly troops in a specific range of locations.

1. The first area of interest is matching all friendly units. Friendly units are of *type* "a-f" (atom-friendly). They can have further classification, such as Ground or Air, but all friendlies are requested. The test "match" allows the use of regular expressions, which are beneficial for the open-ended feature of the *type* category, which is the field that will be tested. So this test will look like: type = match(a-f.*). The ".*" following the "a-f" is a reg exp. The period represents any single character, while the asterisk means zero of more of the proceeding element. This yields a test for all *types* that start with "a-f" and followed by anything.

2. The second test should limit messages to only those in a specific area. There are two ways to do this: using the cylindrical error field of a position test (section 7.2) or by checking whether the message's "point.le" and "point.lon" are within range. The two choices offer different pros and cons, such as a circular area of interest or a square area of interest. Because the blue force tracking map is rectangular, we'll opt for the latter option. Because there are two fields, it will require two separate tests. The finals tests would look like "point.le = n_range(x,y)" and "point.lon = n_range(j,k)", where x,y,j,k are different numbers. These check to see if the points are within the numerical range desired.



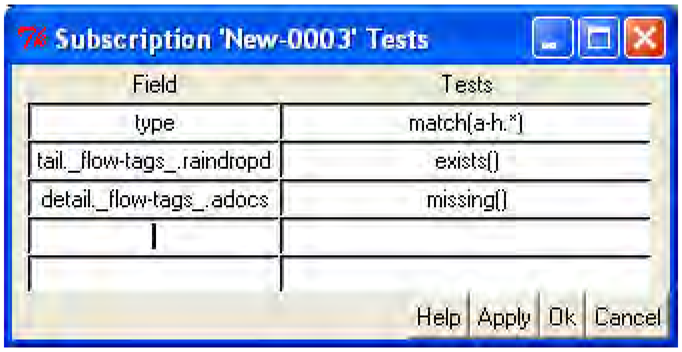
Field	Tests
type	match(a-f.*)
point.le	n_range(x,y)

Figure 7-1. Example 1 Tests

7.2 RainDrop and ADOCS

RainDrop is a targeting system that uses stereo imagery to simulate 3D landscapes, which yields precise coordinates of targets. It is made to “speak” CoT by the plug-in program raindropd. ADOCS, among other abilities, acts as a target approval system, allowing the required approvals to be gotten quickly, which leads to faster responses and strikes. Say a stationary unit wishes to get all approved mensurated targets from RainDrop within ~1000 nautical miles. The Cursor-on-Target Message Router would then have to check for all messages in this range and if they were mensurated by raindropd sent to ADOCS for approval, or if approved by ADOCS rerouted to the stationary unit. This will require two different subscriptions.

1. Some tests will be common to both subscriptions, namely, a positional test, an atomic test for hostiles, and a test that the message was created via RainDrop. The positional test is easy, and should be in the Edit Event window, not the Edit Test window. The point.lat, point.lon, and point.hae are set to the position of the stationary unit. The cylindrical error, point.ce, is set to 1852000 meter (1000 NM). The linear error, point.le, doesn’t matter so it is kept at 99999999.
2. The next test is for hostility, and is completed exactly like step one in the previous example, but changing friendly for hostile. The test looks like: type = match(a-h.*).
3. To test for RainDrop, we’ll use the CoT XML sub-schema flow-tags. Flow tags, hierarchically under detail, show what systems have touched the CoT message and at what time. If a message was mensurated with RainDrop, it will have a flow-tag “raindropd” (because raindropd actually creates the XML message). To check the existence of that flow-tag, we’ll use the test “exists().” The test will look like: detail._flow-tags_.raindropd = exists().
4. The differences between the two subscriptions come in testing for ADOCS approval. If a message has gone through ADOCS, it will also have a flow-tag of “adocs.” Using the same field (detail._flow-tags_.adocs), but with different tests, will finish the subscriptions. The test “missing()” returns true if the field in question isn’t present, so any message needing ADOCS approval will have the test: detail._flow-tags_.adocs = missing(). Any message already through ADOCS will have the test: detail._flow-tags_.adocs = exists().



Subscription 'New-0003' Tests

Field	Tests
type	match(a-h.*)
tail._flow-tags_.raindropd	exists()
detail._flow-tags_.adocs	missing()

Figure 7-2. Example 2 Tests Before ADOCS

Appendix A

Is Predicate Tests

The following table is a list of the possible entries that can be tested against using “Is()”. The “Is” test is typically used against the CoT *type* field, though there are other fields to test against, namely *how* and *qos*. The first column is the test entry, while the second column shows the regular expression that it corresponds to, were you using the “match()” test. There are further subsections below with descriptive headers.

Test	Reg Exp	Test	Reg Exp
'friend'	="^a-f-"	'bits.friend'	="^b-(g|r)-f-"
'friendly'	="^a-f-"	'bits.friendly'	="^b-(g|r)-f-"
'hostile'	="^a-h-"	'bits.hostile'	="^b-(g|r)-h-"
'unknown'	="^a-u-"	'bits.unknown'	="^b-(g|r)-u-"
'pending'	="^a-p-"	'bits.pending'	="^b-(g|r)-p-"
'assumed'	="^a-a-"	'bits.assumed'	="^b-(g|r)-a-"
'neutral'	="^a-n-"	'bits.neutral'	="^b-(g|r)-n-"
'suspect'	="^a-s-"	'bits.suspect'	="^b-(g|r)-s-"
'joker'	="^a-j-"	'bits.joker'	="^b-(g|r)-j-"
'faker'	="^a-k-"	'bits.faker'	="^b-(g|r)-k-"
'unknown'	="^a-u-"	'bits.unknown'	="^b-(g|r)-u-"
'atoms'	="^a-"	‘bits'	="^b-"
'airborne'	="^a-.-A"	'strikewarn'	="^b-S"
'air'	="^a-.-A"	'cuepoint'	="^b-m-p-s-p-i"
'ground'	="^a-.-G"	'click'	="^b-m-p-m-c"
'installation'	="^a-.-G-I"	'spi'	="^b-m-p-s-p-i"
'vehicle'	="^a-.-G-E-V"	'refpoint'	="^b-m-p"
'equipment'	="^a-.-G-E"	'grid'	="^b-m-g-o"

Test	Reg Exp	Test	Reg Exp
'gnd'	="^a-.-G"	'tacelint'	="^b-d-r"
'surface'	="^a-.-S"	'image'	="^b-i"
'sea'	="^a-.-S"	'subscription'	="^t-b"
'sam'	="^a-.-A-W-M-S"	'mootw'	="^b-r-.-O"
'subsurface'	="^a-.-U"	‘any’	=".*"
'sub'	="^a-.-U"	‘true’	=".*"
'uav'	="^a-f-A-M-F-Q-r"	'false'	="^$"
Tasking or request types		'report'	="^b-r-"
'tasking'	="^t-"	'weather'	="^b-w"
't.remarks'	="^t-x-f"	'graphic'	="^b-g-"
't.state'	="^t-x-s"	**Machine-to-machine request-response processing**	
't.sync'	="^t-x-s"	'reply'	="^y-"
't.isrreq'	="^t-s"	'r.complete'	="^y-c"
't.cancel'	="^t-z"	'r.success'	="^y-c-s"
't.commcheck'	="^t-x-c-c"	'r.fail'	="^y-c-f"
't.dgps'	="^t-x-c-g-d"	'r.failed'	="^y-c-f"
't.strike'	="^t-k"	'r.ack'	="^y-a"
't.destroy'	="^t-k-d"	'r.receipt'	="^y-a-r"
't.investigate'	="^t-k-i"	'r.wilco'	="^y-a-w"
't.target'	="^t-k-t"	'r.executing'	="^y-s-e"
These are compared against the "how" field		'r.rejected'	="^y-c-f-r"
'h.mensurated'	="^m-i"	'r.stale'	="^y-c-f-s"
'h.human'	="^h"	'r.review'	="^y-s-r"
'h.retyped'	="^h-t"	'r.completion'	="^y-c"
'h.machine'	="^m"	**QoS predicates**	

Test	Reg Exp	Test	Reg Exp
'h.gps'	="^m-g"	'q.guaranteed'	="^.-.-g"
'h.nonCoT'	="^h-g-i-g-o"	'q.assured'	="^.-.-g"
'h.gigo'	="^h-g-i-g-o"	'q.deadline'	="^.-.-d"
'h.estimated'	="^h-e"	'q.congestion'	="^.-.-c"
'h.calculated'	="^h-c"	'q.low'	="^[0-3]-.-."
'h.transcribed'	="^h-t"	'q.med'	="^[4-6]-.-."
'h.pasted'	="^h-p"	'q.high'	="^[7-9]-.-."
'h.magnetic'	="^m-m"	'q.routine'	="^[0-1]-.-."
'h.ins'	="^m-n"	'q.priority'	="^[2-3]-.-."
'h.ins-gps'	="^m-g-n"	'q.immediate'	="^[4-5]-.-."
'h.simulated'	="^m-s"	'q.flash'	="^[6-7]-.-."
'h.configured'	="^m-c"	'q.flashover'	="^[8-9]-.-."
'h.radio'	="^m-r"	'q.replace'	="^.-r-."
'h.passed'	="^m-p"	'q.follow'	="^.-f-."
'h.fused'	="^m-f"	**Operational predicates**	
'h.tracker'	="^m-a"	'o.exercise'	="^e-"
'h.ins+gps'	="^m-g-n"	'o.operation'	="^o-"
'h.dgps'	="^m-g-d"	'o.simulated'	="^.-s-"
'h.eplrs'	="^m-r-e"	'o.simulation'	="^.-s-"
'h.plrs'	="^m-r-p"		
'h.doppler'	="^m-r-d"		
'h.vhf'	="^m-r-v"		
'h.tadil'	="^m-r-t"		
'h.tadila'	="^m-r-t-a"		
'h.tadilb'	="^m-r-t-b"		

Test	Reg Exp	Test	Reg Exp
'h.tadilj'	="^m-r-t-j"		
'h.nonCoT'	="-g-i-g-o"		